BAB 1

MENGENAL JAVA

Bagian awal ini akan mengajak Anda untuk mengenal lebih dekat bahasa pemrograman Java. Lebih khusus lagi, kita akan mengeksplorasi komponen-komponen fundamental yang perlu sekali diketahui dan dipahami dengan baik. Selain itu, bagian ini juga akan mengulas secara garis besar tentang fitur-fitur baru Java versi 5.0 dan 6. Diharapkan uraian ini nantinya dapat membantu memudahkan Anda ketika ingin mengungkap rahasia-rahasia Java selanjutnya.

1 Sekilas Java

Tentunya Anda tidak asing lagi dengan nama Java, sebuah bahasa pemrograman berorientasi objek yang dikembangkan oleh Sun Microsystems. Di bagian awal ini, kita akan mengulas tentang sekilas Java yang ditinjau dari aspek lingkungan pengembangan dan produk. Diharapkan ulasan ini nantinya dapat memperjelas terminologi ataupun pernyataan-pernyataan yang kerap kali membingungkan, terutama bagi yang baru mengenal Java.

- Lingkungan Pengembangan

Dalam mendiskusikan Java, kiranya penting sekali untuk membedakan antara bahasa pemrograman Java, Java Virtual Machine, dan platform Java. Bahasa pemrograman Java adalah bahasa yang digunakan untuk menghasilkan aplikasi-aplikasi Java. Pada umumnya, bahasa pemrograman hanya mendefinisikan sintaks dan perilaku bahasa.

Pada saat program Java dikompilasi, ia akan dikonversi ke bentuk bytecode, yang merupakan bahasa mesin yang portable. Selanjutnya, bytecode tersebut dijalankan di Java Virtual Machine (atau disebut Java VM atau JVM). Meskipun JVM dapat diimplementasikan langsung di perangkat keras, namun biasanya diimplementasikan dalam bentuk program perangkat lunak yang mengemulasi mesin (komputer) dan digunakan untuk menginterpretasi bytecode.

Platform dapat didefinisikan sebagai perangkat lunak pendukung untuk aktivitas-aktivitas tertentu. Platform Java sendiri pada prinsipnya berbeda dengan bahasa Java atau JVM. Platform Java adalah himpunan kelas-kelas Java yang sudah didefinisikan sebelumnya dan eksis sejak instalasi Java. Platform Java juga mengacu pada lingkungan runtime atau API (*Application Programming Interface*) Java.

- Edisi Java

Guna mencakup lingkungan-lingkungan aplikasi yang berbeda, Sun mendefinisikan 3 (tiga) edisi Java.

  - J2ME (Java 2 Micro Edition)

    Edisi ini ditujukan bagi lingkungan dengan sumber daya terbatas, seperti smartcard, ponsel, dan PDA.

  - J2SE (Java 2 Standard Edition)

    Edisi ini ditujukan bagi lingkungan workstation, seperti pembuatan aplikasi-aplikasi dekstop.

  - J2EE (Java 2 Enterprise Edition)

    Edisi ini ditujukan bagi lingkungan Internet atau aplikasi terdistribusi dalam skala besar.

Perbedaan setiap edisi meliputi fitur-fitur bahasa yang didukung dan API yang tersedia. Berdasarkan tingkatannya, edisi yang lebih tinggi mampu mengemulasikan edisi yang lebih rendah. Adapun urutan edisi dari yang tertinggi ke rendah adalah J2EE, J2SE, dan J2ME.

- Versi Java

Ada hal yang menarik dan perlu kita cermati mengenai versi-versi Java yang telah dirilis. Sun menggunakan dua jenis versi untuk mengidentifikasikan rilis Java, yaitu versi produk dan versi developer. Seperti kita ketahui, versi terbaru saat ini adalah versi 6 (versi produk) atau versi 1.6.0 (versi developer), dengan nama kode Mustang.

Sejak tahun 2006, Sun juga menyederhanakan penamaan platform dengan tujuan untuk mencerminkan tingkat kematangan, stabilitas, skalabilitas, dan sekuriti yang lebih baik. Jika penamaan versi sebelumnya adalah Java 2 Platform, Standard Edition 5.0 (J2SE 5.0), maka sekarang disederhanakan menjadi Java Platform, Standard Edition 6 (Java SE 6, atau lebih sering disebut Java 6).

2 Kompilasi dan Interpretasi

Seperti diketahui, Java adalah bahasa pemrograman yang kode programnya dikompilasi dan diinterpretasi. Meskipun pembuatan aplikasi Java dapat dilakukan melalui IDE (*Integrated Development Environment*), namun di sini kita memfokuskan pada tool command-line untuk kompilasi dan interpretasi.

- Kompilasi

Kompilasi kode program Java dilakukan menggunakan tool command-line yang bernama **javac**, atau biasa disebut kompiler Java. Tahap kompilasi ini bertujuan untuk mengonversi kode sumber ke program biner yang berisi bytecode, yaitu instruksi-instruksi mesin. Contoh berikut memperlihatkan cara melakukan kompilasi pada file program Coba.java (asumsi sudah berada di command-line atau shell).

```
javac Coba.java
```

Saat mengompilasi kode program, kita juga diperkenankan untuk menspesifikasikan versi rilis tertentu. Aturan dasar dalam spesifikasi versi ini cukup sederhana, di mana versi terbaru dapat mengenali versi-versi di bawahnya, namun tidak demikian sebaliknya. Sebagai contoh, untuk mengetahui apakah kode program dapat berjalan di versi 1.4 ataukah tidak, tambahkan opsi **–source 1.4**.

```
javac –source 1.4 Coba.java
```

Jika **–source** digunakan untuk menspesifikasikan rilis asal, opsi **–target** berfungsi untuk menetapkan versi tujuan. Opsi-opsi lain yang sering digunakan diperlihatkan sebagai berikut:

```
// Menetapkan lokasi file-file kelas (classpath)
javac -cp D:\java Coba.java
javac -classpath D:\java Coba.java

// Menetapkan lokasi file .class yang akan dihasilkan
javac -d D:\java Coba.java
// Hasil: file Coba.class diletakkan di D:\java

// Mendapatkan informasi mengenai apa yang dilakukan kompiler
javac -verbose Coba.java

// Mendapatkan informasi versi (developer)
javac -version
```

Sekadar catatan, untuk memudahkan pemanggilan kompiler, tambahkan path yang berisi file-file executable (di direktori bin) ke variabel sistem Path. Untuk lebih praktisnya, Anda bisa menggunakan kotak dialog **Environment Variables** (melalui **System Properties**).

Apabila Anda bekerja di lingkungan Unix/Linux, modifikasilah file **/etc/profile** dengan menambahkan baris berikut:

```
PATH=/lokasi_instalasi/bin:$PATH
export PATH
```

- Interpretasi

Sebagaimana disinggung, kode program Java tidak dieksekusi di komputer secara langsung, tetapi berjalan di atas komputer hipotesis yang distandardisasikan, yang disebut Java Virtual Machine. Untuk menginterpretasi bytecode, kita menggunakan tool bernama **java**, atau biasa disebut interpreter Java. Pada saat menginterpretasi, Anda

tidak perlu menyertakan ekstensi file (.java atau .class), cukup nama file saja.

```
java Coba
```

Untuk kasus program-program berbasis teks, hasil keluaran akan langsung ditampilkan di command-line. Terkait hal ini, tool java memungkinkan Anda untuk meng-capture hasil keluaran dan menyimpannya di sebuah file.

Contoh perintah berikut akan menangkap hasil keluaran program Coba dan menyimpannya di file **coba.txt**.

```
java Coba > coba.txt
```

Apabila Anda menggunakan perintah di atas pada aplikasi GUI, maka file keluaran akan tetap diciptakan, namun tidak ada isinya (dengan asumsi bahwa program tidak mencetak teks keluaran).

3 Elemen Bahasa

Secara garis besar, elemen-elemen di setiap bahasa pemrograman sebenarnya hampir sama. Meskipun demikian, ada elemen-elemen khusus yang membedakan dan sekaligus mencerminkan identitas suatu bahasa. Adapun mengingat di sini kita bekerja dengan bahasa pemrograman Java, tentunya kita juga perlu memahami elemen-elemen dasar bahasa ini.

- Tipe Data

Tipe data di Java dikelompokkan menjadi dua jenis, yaitu tipe primitif dan reference (kelas). Tipe primitif/dasar adalah tipe-tipe bawaan, meliputi `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, dan `double`. Sementara itu, tipe reference memiliki semantik seperti pointer. Jenis tipe reference meliputi kelas, interface, dan array.

Apabila Anda ingin memperlakukan nilai primitif sebagai suatu objek, Anda bisa memanfaatkan kelas-kelas pembungkus (wrapper). Kelas-kelas tersebut meliputi `Boolean`, `Character`, `Byte`, `Short`, `Integer`, `Long`, `Float`, dan `Double`. Perhatikan sintaksnya (penulisan huruf kecil dan besar), agar tidak keliru dengan tipe-tipe primitif.

- Deklarasi dan Inisialisasi Variabel

Sebelum digunakan, setiap variabel harus dideklarasikan terlebih dahulu. Langkah ini dilakukan dengan menetapkan tipe data dan nama variabel. Pernyataan deklarasi variabel tunggal juga dapat digunakan untuk mendeklarasikan lebih dari satu variabel, tetapi semuanya harus bertipe sama.

```
int i;
int j;

// ekuivalen dengan kedua pernyataan di atas
int i, j;
```

Deklarasi juga dapat sekaligus melakukan inisialisasi terhadap suatu variabel.

```
int i = 2;
int j = 3;

int i = 2, j = 3;
```

- Initial Value

Saat mendeklarasikan variabel kelas, kita tidak harus melakukan inisialisasi karena kompiler akan meng-assign initial value (nilai awal atau default). Nilai default untuk semua tipe reference adalah null. Nilai default tipe primitif `boolean` adalah `false`, `char` adalah \u0000, integer (`byte`, `short`, `int`, `long`) adalah 0, dan floating point (`float`, `double`) adalah 0.0.

- Ruang Lingkup Variabel

Java mendefinisikan empat jenis variabel, meliputi variabel instance (field non-statis), variabel kelas (field statis), variabel lokal, dan parameter. Istilah field mengacu pada variabel instance dan variabel kelas (terkadang disebut member variable). Sementara itu, istilah variabel mengacu pada semua jenis variabel.

Lokasi di mana suatu variabel dideklarasikan secara eksplisit juga menetapkan ruang lingkupnya. Ruang lingkup variabel adalah wilayah di mana suatu variabel dapat diacu melalui namanya. Ruang lingkup juga menyatakan kapan variabel akan diciptakan dan dihapus dari memori.

- Blok

Blok adalah kelompok pernyataan (nol atau lebih) di dalam tanda kurung kurawal. Penggunaan blok dengan pernyataan alir kontrol sangat direkomendasikan, meskipun hanya melibatkan sebuah pernyataan.

```
if (kondisi)
{ // awal blok

     // pernyataan

} // akhir blok
```

- Komentar

Penulisan komentar dapat mengadopsi blok komentar gaya C ataupun C++. Komentar gaya bahasa C lazimnya digunakan untuk komentar yang terdiri atas beberapa baris. Sementara itu, komentar gaya C++, yang dinyatakan melalui karakter //, umumnya digunakan untuk komentar satu baris.

Untuk komentar-komentar yang akan dimasukkan ke dokumentasi dan dihasilkan melalui tool javadoc, disarankan menggunakan /** dan diakhiri dengan karakter */.

4 Aturan Penamaan

Di dalam pemrograman, suatu nama digunakan untuk mengacu ke entitas yang dideklarasikan. Terkait hal ini, ada beberapa aturan dasar penamaan yang perlu sekali diperhatikan dalam upaya menghasilkan kode program yang *readable*.

- Penamaan Paket

Nama awal paket sebaiknya terdiri atas dua atau tiga huruf kecil, dan biasanya menggunakan nama domain Internet, seperti com, org, net, dan edu. Selain itu, Anda juga diperkenankan memberi nama paket dengan kode-kode negara, seperti id, uk, atau au. Penggunaan nama domain ini bertujuan untuk mencegah terjadinya konflik paket, dengan asumsi bahwa Anda tidak menggunakan nama domain orang lain. Sebagai contoh, nama paket berbasis domain http://didik.indodesain.com adalah `com.indodesain.didik`.

- Penamaan Kelas dan Interface

Nama kelas dan interface sebaiknya berupa kata benda atau ungkapan kata benda yang deskriptif dan tidak terlalu panjang. Penulisan nama mengacu pada sintaks Pascal, di mana huruf pertama untuk setiap kata adalah huruf besar dan tidak ada spasi, misalnya `Bangun`, `SegiTiga`, atau `KoneksiData`.

- Penamaan Method

Nama method seharusnya berupa kata kerja atau ungkapan kata kerja. Penulisan method mengacu pada sintaks Camel, di mana huruf pertama untuk setiap kata pertama adalah huruf kecil dan huruf pertama kata selanjutnya adalah huruf besar. Nama method umumnya juga mencerminkan operasi yang dilakukannya, contohnya seperti `setData`, `getData`, `isValidData`, atau `toString`.

- Penamaan Variabel

Penamaan variabel-variabel kelas (*fields*) mirip dengan penamaan method. Untuk penamaan variabel lokal dan parameter, seringkali menggunakan suatu akronim, singkatan, atau istilah-istilah yang mudah diingat, contohnya seperti sr (`StreamReader`), buf (buffer), d (`double`), dan s (`String`).

- Penamaan Konstanta

Seperti umumnya bahasa pemrograman, nama konstanta di Java harus berupa huruf besar semua. Apabila nama konstanta terdiri atas beberapa kata, sebaiknya pisahkan dengan tanda garis bawah "_". Contoh penamaan konstanta misalnya `MAX`, `MAX_DATA`, atau `MAX_LEN_DATA`.

5 Paket dan Namespace

Paket bertujuan mengorganisir keterhubungan kelas dan mendefinisikan namespace untuk kelas-kelas yang berada di dalamnya. Dalam upaya memudahkan penggunaan, menghindari konflik nama, dan mengontrol akses kelas-kelas maupun interface-interface, kita bisa mengelompokkannya ke dalam suatu paket.

- Deklarasi Paket

Paket dideklarasikan menggunakan pernyataan `package` yang diikuti dengan nama paket dan subpaket (jika ada). Deklarasi paket sebaiknya dilakukan di bagian paling awal kode program. Contoh deklarasi paket diperlihatkan seperti berikut:

```
package com.indodesain.didik;
```

Dalam implementasinya, nama paket dan subpaket sebenarnya mencerminkan struktur direktori dengan susunan sesuai penamaan. Apabila kita tidak menggunakan paket, maka kelas terkait merupakan bagian dari paket default (tanpa nama).

- Mengakses Member Paket

Kelas-kelas dan interface-interface di dalam paket, atau disebut member paket, hanya dapat diakses dari luar paket apabila ia didefinisikan sebagai `public`. Ada dua pendekatan yang bisa kita gunakan untuk mengakses member paket, yaitu dengan mengacu nama lengkap dan mengimpor member.

Misalkan di subpaket didik terdapat kelas `Test`, maka cara mengacunya adalah seperti berikut:

```
com.indodesain.didik.Test t = new com.indodesain.didik.Test();
```

Untuk pendekatan kedua, kita terlebih dahulu menuliskan keyword `import` yang diikuti nama paket.

```
import com.indodesain.didik.Test;

...

// Instantiasi di body kelas
Test t = new Test();
```

Pada pendekatan kedua, Anda juga diperkenankan mengimpor seluruh member yang dinyatakan dengan karakter asterik (*).

```
import com.indodesain.didik.*;
```

Walaupun pendekatan acuan dengan nama lengkap terkesan kurang efektif, namun dalam situasi-situasi tertentu sangat diperlukan. Sebagai contoh, pendekatan ini lazim digunakan untuk menghindari konflik ketika mengakses member di beberapa paket dengan nama sama.

- Impor Otomatis

Secara otomatis JRE akan mengimpor member yang ada di paket `java.lang`, paket default, dan *current* paket. Oleh karena itu, pada saat menggunakan member di paket-paket tersebut, kita tidak perlu melakukan impor secara eksplisit. Sebagai contoh, kita bisa langsung menggunakan kelas `String` tanpa terlebih dahulu mengimpor `java.lang.String` ataupun mengacu nama lengkapnya.

Di Java 5.0, ada fitur baru yang memungkinkan kita untuk mengimpor dengan tambahan pernyataan `static`. Penjelasan mengenai fitur ini akan kita bahas lebih lanjut setelah mengulas pernyataan `static`.

- Strategi Impor Paket

Seringkali kita melihat program-program Java yang mendeklarasikan pernyataan `import` untuk mengakses member paket secara lengkap. Contohnya seperti berikut:

```
import java.io.File;
import java.io.FileReader;
import java.net.URL;
```

Tak jarang pula, kita melihat deklarasi yang menggunakan karakter asterik. Contohnya seperti berikut:

```
import java.io.*;
import java.net.*;
import java.util.*;
```

Sebenarnya, apa kelebihan dan kekurangan masing-masing pendekatan di atas? Apakah pengaksesan member paket dengan nama lengkap lebih cepat dieksekusi dibanding penggunaan karakter asterik?

Pada prinsipnya, deklarasi lengkap memang lebih cepat dikompilasi dibanding pendekatan asterik. Meski demikian, deklarasi lengkap tidak menawarkan kelebihan ketika program dieksekusi (diinterpretasi). Dalam segi efisiensi penulisan, pendekatan asterik tentu menawarkan kelebihan bagi kita. Selain itu, ketika kita ingin mengganti penggunaan kelas (misal dalam satu paket), kita tidak perlu menghapus deklarasi sebelumnya dan mengganti dengan deklarasi baru.

Meskipun pengaksesan dengan karakter asterik tidak berpengaruh terhadap eksekusi program, namun bukan berarti pendekatan ini paling efisien. Setidaknya, kita bisa menggunakan strategi berdasarkan kasus yang kita hadapi. Sebagai contoh, jika kita hanya memerlukan satu atau dua kelas/interface di satu paket, akan lebih efisien jika kita mendeklarasikan nama member paket secara lengkap. Sebaliknya, jika jumlah kelas atau interface yang kita perlukan cukup banyak, tentu akan praktis jika kita menggunakan karakter asterik.

6 Kelas

Kelas merupakan salah satu konsep fundamental pemrograman berorientasi objek. Kelas dapat diilustrasikan sebagai suatu cetak biru (*blueprint*) atau prototipe yang digunakan untuk menciptakan objek. Terkait dengan konsep penting ini, ada beberapa subbahasan yang akan kita ulas di sini.

- Definisi Kelas

Definisi kelas terdiri atas dua komponen, yaitu deklarasi kelas dan body kelas. Deklarasi kelas adalah baris pertama di suatu kelas, dan minimal mendeklarasikan nama kelas. Sementara itu, body dideklarasikan setelah nama kelas dan berada di antara kurung kurawal.

```
// deklarasi kelas
public class ContohKelas {

  // body kelas

}
```

Di Java, nama kelas sekaligus merepresentasikan nama file kode program, dan sifatnya adalah case-sensitive.

- Konstruktor

Kegunaan utama konstruktor adalah untuk menetapkan status awal manakala objek diciptakan. Ada beberapa catatan penting yang harus kita perhatikan dalam pendeklarasian konstruktor. Pertama, nama konstruktor harus sama dengan nama kelas. Kedua, konstruktor boleh memiliki argumen lebih dari satu atau tidak sama sekali. Ketiga, konstruktor tidak boleh mengembalikan suatu nilai.

```
public class ContohKelas {

  // Konstruktor
  public ContohKelas() {

  }

}
```

Konstruktor yang tidak memiliki argumen dinamakan sebagai konstruktor default. Apabila suatu kelas dideklarasikan tanpa adanya sebuah konstruktor, maka secara implisit Java akan menambahkan konstruktor default.

- Access Modifier

Kelas dan interface memiliki batasan akses (*access modifier*) yang menyatakan level aksesnya. Apabila kelas dideklarasikan sebagai `public`, maka ia dapat diakses dari mana saja. Jika batasan akses tidak diberikan, kelas tersebut dinamakan *default accessibility*, dan hanya dapat diakses dari dalam paket terkait (tidak termasuk sub-paket).

Batasan akses lainnya untuk kelas level atas adalah `abstract` (tidak dapat diinstantiasi) dan `final` (tidak dapat diperluas). Anda tidak diperkenankan mendeklarasikan kelas level atas sebagai `private` ataupun `protected`.

- Kelas Bersarang

Suatu kelas boleh mendeklarasikan kelas baru di dalamnya, atau biasa disebut *inner class*, atau kelas bersarang. Apabila diperlukan, inner class juga dapat memiliki kelas lagi di dalamnya.

```
public class ContohKelas {

    public class KelasBersarang {
        // body kelas KelasBersarang
    }

}
```

Inner class juga dapat dideklarasikan secara lokal, yaitu di dalam body method. Kelas seperti ini dinamakan *local inner class* atau *local nested class*.

```
public void test() {

    class KelasDiMethod {
```

```
        // body kelas KelasDiMethod
    }

}
```

- Keyword this dan super

Keyword `this` dapat digunakan untuk merepresentasikan suatu *current* objek dan mengakses variabel-variabel kelas serta method.

```
public class KelasInduk {
    int j;

    public void setNilai(int i) {
        // this ini mengacu pada objek KelasInduk
        this.j = i;
    }
}
```

Keyword `super` digunakan untuk mengakses member kelas yang diturunkan (kelas induk).

```
// KelasAnak memperluas/mewarisi KelasInduk
class KelasAnak extends KelasInduk {

    public KelasAnak() {
        // Mengakses method di kelas induk
        super.setNilai(3);
    }

}
```

7 Method

Seperti halnya kelas, ada dua bagian utama dalam definisi method, yaitu deklarasi dan body method. Deklarasi method mendefinisikan semua atribut method, seperti level akses, tipe kembalian (jika ada), dan argumen-argumen (jika ada).

- Method main

Method `main` merupakan method khusus yang berperan sebagai entry point pada aplikasi. Setiap kelas di suatu aplikasi boleh memiliki method `main`, namun hanya satu yang ditetapkan untuk dieksekusi saat aplikasi dijalankan.

```
public static void main(String[] args) {
    // isi method main
}
```

Method `main` harus didefinisikan sebagai `public`, `static`, tidak mengembalikan suatu nilai (`void`), dan memiliki argumen berupa array string. Apabila interpreter tidak menemukan method `main` di suatu aplikasi, akan muncul pesan kesalahan yang diakibatkan tidak adanya entry point.

Sebenarnya, Anda juga bisa mendefinisikan method utama dengan keyword `static` saja. Pendefinisian seperti ini terlihat tidak mencerminkan struktur method seperti pada umumnya.

```
public class ContohKelas {

  static {
    System.out.println("Halo Indonesia");
      System.exit(0);
  }

}
```

Meskipun kode program di atas dapat berjalan seperti yang diharapkan, namun pendekatan tersebut tidak lazim digunakan.

- Modifier Method

Modifier dari sebuah method dapat terdiri atas nol atau lebih keyword modifier, seperti `public`, `protected`, atau `private`. Keberadaan modifier ini membantu kelas untuk mendefinisikan suatu kontrak, sehingga client dapat mengetahui layanan-layanan yang ditawarkan oleh kelas.

Selain beberapa keyword modifier di atas, yang umumnya sudah kita pahami kegunaannya, ada beberapa modifier lain yang bisa kita spesifikasikan. Salah satu keyword modifier yang sering digunakan adalah `static`. Modifier ini mengizinkan kita untuk mengakses method tanpa perlu menginstantiasi kelas yang mendefinisikannya. Sebagai gantinya, sebelum memanggil method, kita terlebih dahulu menuliskan nama kelas terkait.

```
class Test {
    public static void sayHello() {
        System.out.println("hello");
    }

    public void sayNo() {
        System.out.println("no");
    }
}
```

```
public class AksesStatis {
    public static void main(String[] args) {
        // Akses method statis
        Test.sayHello();

        // Error, akses method non-statis
        // Test.sayNo();
        // Harus begini
        Test t = new Test();
        t.sayNo();
    }
}
```

- Variabel Lokal

Sebelum variabel lokal dapat digunakan, ia harus diinisialisasi terlebih dahulu. Kondisi ini berbeda dengan variabel kelas, di mana secara otomatis akan diinisialisasi. Penggunaan variabel lokal yang tanpa diinisialisasi akan mengakibatkan kesalahan pada saat kompilasi.

- Overloading Method

Overloading method adalah kemampuan untuk mendefinisikan beberapa method di sebuah kelas dengan nama sama. Aturan dasar overloading adalah jumlah atau tipe argumen harus berbeda. Apabila jumlah dan tipe argumen sama, maka urutannya harus berbeda.

```
int Test() {
    return 1;
}

int Test(int a) {
    return a;
}

int Test(double a, int b) {
    return b;
}

int Test(int i, double j) {
    return i;
}

// Ini akan error, sudah didefinisikan di method sebelumnya
void Test(int x, double y) {
}
```

8 Objek

Di pemrograman berorientasi objek, objek adalah entitas dasar saat runtime. Pada saat kode program dieksekusi, objek berinteraksi satu sama lain tanpa harus mengetahui detil data atau kodenya. Interaksi antara objek ini dilakukan menggunakan suatu *message*. Objek memiliki suatu siklus hidup, yaitu diciptakan, dimanipulasi, dan dihancurkan.

- Menciptakan Objek

Objek diciptakan menggunakan operator `new`. Dari sisi kelas, langkah ini merupakan instantiasi kelas. Selanjutnya objek yang berhasil diciptakan tersebut akan diletakkan di memori heap.

```
ContohKelas ck = new ContohKelas();
```

Dalam kasus-kasus tertentu, terkadang kita juga dapat menciptakan objek tanpa harus meng-assign ke variabel. Langkah ini umumnya dilakukan apabila kita tidak memerlukan referensi ke objek tersebut. Sebagai contoh, jika kita memiliki method yang menerima argumen berupa objek `ContohKelas`, maka dapat kita tuliskan seperti berikut:

```
getData(new ContohKelas());
```

- Memeriksa Tipe Objek

Anda bisa memanfaatkan fungsionalitas operator `instanceof` untuk mengetahui tipe suatu objek pada saat runtime. Operator ini akan mengembalikan nilai `true` apabila tipe objek sesuai, sebaliknya mengembalikan nilai `false`.

```
ContohKelas ck = new ContohKelas();
System.out.println(ck instanceof ContohKelas);
// Output: true
```

Perlu diperhatikan, `instanceof` akan selalu mengembalikan nilai `false` jika variabel objek diinisialisasi dengan nilai `null`. Ini karena nilai `null` tidak mencerminkan objek apa pun.

```
ContohKelas ck2 = null;
System.out.println(ck2 instanceof ContohKelas);
// Output: false
```

Operator `instanceof` hanya dapat digunakan pada tipe reference dan objek. Penggunaan operator ini pada tipe primitif akan mengakibatkan kesalahan saat kompilasi.

- Menghapus Objek

Java menggunakan teknik yang dikenal sebagai garbage collection untuk menghapus objek-objek yang sudah tidak diperlukan. Dengan demikian, kita tidak perlu khawatir akan terjadinya kebocoran memori. Dalam praktiknya, garbage collector mampu mengidentifikasi kapan suatu objek dialokasikan dan kapan ia tidak digunakan lagi.

Garbage collector melakukan tugasnya secara tak sinkron berdasarkan ketersediaan sumber daya. Normalnya, jika suatu objek sudah tidak diacu (di-refer), maka ia akan segera dibersihkan. Terlepas dari mekanisme normal ini, kita juga dapat memanggil garbage collector secara eksplisit menggunakan method statis `gc`.

```
System.gc();
```

Perlu sekali diperhatikan, tidak semua jenis objek akan ditangani oleh garbage collector. Untuk objek-objek eksternal, seperti file dan database, sebaiknya kita tangani secara eksplisit.

9 Exception Handling

Eksepsi (*exception*) adalah suatu even, yang terjadi selama eksekusi program, yang mengacaukan alir normal instruksi program. Pada prinsipnya, eksepsi adalah suatu objek, yang diturunkan dari kelas `java.lang.Throwable`. Dalam menangani suatu eksepsi, Java menggunakan mekanisme penanganan eksepsi terstruktur.

- Menangkap Eksepsi

Ada dua jenis blok kode yang dapat kita gunakan untuk menangani eksepsi, yaitu `try` dan `catch`. Blok `try` berisi kode yang berpotensi membangkitkan eksepsi, sedangkan blok `cath` merupakan exception handler-nya.

```
int i = 10;
int j = 0;

try {
```

```
    // Baris berikut akan membangkitkan eksepsi, karena
    // pembagian dengan nol, sehingga perlu ditangkap
    int n = i / j;
    // Baris berikut tidak akan dieksekusi
    System.out.println(n);
} catch (Exception ex) {
    System.out.println("Eksepsi ditangkap\n" + ex.getMessage());
}
```

Apabila penanganan eksepsi terdiri atas beberapa blok `catch`, sebaiknya letakkan objek yang paling relevan di blok terdekat. Langkah ini bertujuan agar eksepsi yang terjadi dapat ditangkap oleh blok yang sesuai, dan menjadikan kode program mengalir secara natural.

```
try {
    int n = i / j;
    System.out.println(n);
} catch (ArithmeticException ae) {
    System.out.println("ArithmeticException");
} catch (Exception e) {
    System.out.println("Eksepsi ditangkap");
}
```

Penanganan eksepsi juga dapat melibatkan blok `finally`, yaitu blok yang akan selalu dieksekusi. Blok ini umumnya sering dimanfaatkan untuk tahap pembersihan sumber daya karena sifatnya yang selalu dijalankan.

```
try {
    int n = i / j;
    System.out.println(n);
} catch (ArithmeticException ex) {
    System.out.println("ArithmeticException");
} catch (Exception ex) {
    System.out.println("Eksepsi ditangkap");
} finally {
    System.out.println("Ini akan selalu dieksekusi");
}
```

- Melempar Eksepsi

Untuk menspesifikasikan eksepsi yang akan diperiksa, kita dapat memanfaatkan klausa `throws`.

```
public static int pembagian(int i, int j)
throws ArithmeticException {
    return i/j;
}
```

Klausa `throws` di atas menyatakan bahwa pemanggilan method pembagian harus dilakukan menggunakan blok `try` dan `catch`.

```
try {
    int l = pembagian(2,0);
    System.out.println(l);
} catch (ArithmeticException ex) {
    ex.printStackTrace();
}
```

Kita juga dapat menangkap eksepsi secara eksplisit menggunakan pernyataan `throw` (perhatikan, jangan keliru dengan `throws`).

```
public static int pembagian(int i, int j)
throws ArithmeticException {

    if (j == 0) {
        throw new ArithmeticException("Pembagian dengan 0");
    }
    return i/j;
}
```

- Informasi Eksepsi

Kelas `Throwable` mendefinisikan sejumlah method yang dapat membantu kita untuk mendapatkan informasi-informasi terkait dengan eksepsi, di antaranya adalah method `getMessage` dan `printStackTrace`.

Apabila Anda ingin mendapatkan informasi mengenai method atau nama kelas terkait, gunakan method `getStackTrace` dan objek `StackTraceElement`.

```
try {
    int l = pembagian(2,0);
    System.out.println(l);

} catch (ArithmeticException ex) {
    System.out.println("Nama File: " +
        ex.getStackTrace()[0].getFileName());
    System.out.println("Nama Kelas: " +
        ex.getStackTrace()[0].getClassName());
    System.out.println("Nama Method: " +
        ex.getStackTrace()[0].getMethodName());
    System.out.println("Baris ke-" +
        ex.getStackTrace()[0].getLineNumber());
}
```

10 Fitur Baru Java 5.0

Meskipun Sun Microsystems sudah merilis Java versi 6, namun rasanya belum terlambat jika kita membicarakan fitur-fitur baru Java 5.0. Secara garis besar, ada tujuh fitur utama yang diperkenalkan oleh versi dengan nama kode Tiger ini.

- Tipe Generic

Intuisi dari fitur ini adalah menghasilkan kode yang mudah dideteksi kesalahannya saat kompilasi. Sebagai contoh, di versi sebelumnya, ketika kita hanya ingin menampung string di suatu objek, kompiler tidak akan memprotes meski yang dimasukkan bukanlah string. Kini, Anda dapat menspesialisasi tipe yang awalnya bersifat general.

```
// Mendefinisikan list untuk string
List<String> list = new ArrayList<String>();

// Ini dilaksanakan
list.add("string");

// Ini akan diprotes kompiler (error)
list.add(123);
```

- Tipe Enumerasi

Fitur ini mengizinkan Anda untuk mendeklarasikan tipe enum (enumerasi) dengan mudah. Tak hanya itu, fitur ini juga menyediakan semua keuntungan dari pola Typesafe Enum secara praktis. Dalam implementasinya, deklarasi dilakukan menggunakan keyword `enum`.

```
private static enum NamaHari {
       Minggu, Senin, Selasa, Rabu, Kamis, Jumat, Sabtu
};
```

- Autoboxing/Unboxing

Fitur ini mampu menghilangkan kejenuhan Anda ketika melakukan konversi antara tipe primitif dan pembungkusnya. Seperti diketahui, di versi sebelum Java 5.0, kita harus memperlakukan tipe primitif ke tipe reference (disebut boxing) ketika ingin mengonversi tipe primitif ke pembungkusnya. Sebaliknya, kita melakukan unboxing ketika ingin mengonversi tipe reference ke tipe primitif.

```
// Sebelum Java 5.0
// Konversi primitif ke wrapper (pembungkus)
int i = 3;
Integer box = new Integer(i);
```

```
// Konversi wrapper ke primitif
Integer j = new Integer(200);
int unbox = j.intValue();
```

Sejak Java 5.0, secara otomatis kompiler akan menambahkan kode yang diperlukan untuk melakukan konversi tipe.

```
// Autoboxing/unboxing
// Integer auto di-unbox ke int (tipe primitif),
// kemudian hasil penjumlahan di-boxing ke objek Integer
Integer auto = 3 + 2;
```

- Anotasi

Fitur ini menyediakan suatu cara untuk menghubungkan metadata dengan elemen-elemen program. Java 5.0 mendefinisikan tiga jenis anotasi standard di dalam paket `java.lang`, meliputi `Override`, `Deprecated`, dan `SupressWarnings`. Contoh penggunaan anotasi diperlihatkan seperti berikut:

```
@Override public String toString() {
  return "[" + super.toString() + "]";
}


@Deprecated public static void test() {
  System.out.println("deprecated");
}


// Mengabaikan warning unchecked
@SuppressWarnings(value={"unchecked"})
public static void testSupress() {
  // Kode yang berpotensi mendapat respon
  // warning unchecked di sini
}
```

- Argumen Variabel

Kini Java mendukung argumen array (bukan tipe reference array) melalui fitur varargs. Untuk menggunakan fitur ini, deklarasi tipe pada variabel harus diikuti dengan tanda titik sebanyak tiga kali.

```
  public static void TestVarArgs(String s, int... args) {
    System.out.println("argumen 1= " + s);

    int len = args.length;
    // Ekstraksi argumen
    for (int j=0; j<len; j++) {
      System.out.println("argumen " + (j+2) + "= " + args[j]);
    }
```

```
}
```

Pada saat method di atas dipanggil, kompiler akan menginter-pretasikan sebagai `TestVarArgs(String s, int[] args)`. Meskipun varargs dianggap sebagai array, tetapi kode pemanggil tidak perlu mengirimkan array saat mengisikan argumen.

- Pernyataan for/in

Di beberapa bahasa lain, Anda tentu tidak asing dengan pernyataan `foreach`. Pernyataan seperti ini kini juga dapat kita nikmati di Java 5.0, meskipun namanya bukan foreach.

```
for (NamaHari h : NamaHari.values()) {
    System.out.println(h);
}
```

Ekspresi dengan huruf tebal di atas bisa kita baca "untuk setiap NamaHari h di enumerasi NamaHari". Dari sini terlihat bahwa pen-dekatan for/in dapat menghilangkan kejenuhan dan kesalahan saat melakukan iterasi.

- Impor Static

Penggunaan keyword `import static` memungkinkan Anda untuk mengakses member-member kelas yang sifatnya statis tanpa harus menyertakan nama paket atau kelas.

```
// Tanpa impor static
System.out.println("Halo Indonesia");


// Impor static, dengan terlebih dahulu menuliskan
// import static java.lang.System.out;
// di atas deklarasi kelas
out.println("Halo Indonesia");
```

11 Fitur Baru Java 6

Bagian ini akan menguraikan fitur-fitur utama Java 6 secara garis besar. Dengan demikian, di sini kita tidak akan membahas tentang implementasi fitur terkait. Di bab-bab selanjutnya, kita akan mem-bahas penggunaan fitur baru yang relevan dengan topik bab. Sekilas uraian ini dimaksudkan untuk sekadar memberikan referensi tam-bahan.

- Utilitas dan Elemen Bahasa

Sebagai paket utama, `java.lang` dan `java.util` tak luput dari modifikasi dan penambahan fitur-fitur baru. Fitur baru yang ditambahkan di paket `java.lang` antara lain input/output console dan pemeriksaan string kosong. Di paket `java.util`, juga ditambahkan kelas-kelas dan interface-interface baru, di antaranya interface `Deque` dan `NavigableMap`. Selain itu, paket `java.lang` dan `java.util` juga menambahkan sejumlah method di kelas-kelas dan interface-interface.

- AWT dan Swing

Untuk meningkatkan kemampuan AWT dan Swing, Java menambahkan beragam fitur baru. Di paket AWT, terdapat fitur-fitur menarik seperti splash screen, system tray, modalitas dialog, dan text antialiasing. Adapun untuk menyediakan dukungan yang lebih baik pada aplikasi GUI Swing, ditambahkan fitur pengurutan dan penyaringan tabel, pencetakan di komponen teks, drag dan drop, serta objek `SwingWorker`.

- JDBC 4.0

Kemampuan akses dan manipulasi data melalui aplikasi-aplikasi Java kini semakin ditingkatkan dengan dirilisnya JDBC 4.0. Sejumlah fitur baru yang diperkenalkan antara lain mekanisme loading driver, penanganan eksepsi, fungsionalitas BLOB/CLOB, dukungan karakter nasional, dan anotasi.

- I/O dan Networking

Meskipun perubahannya tidak terlalu signifikan, fitur dan kemampuan paket `java.io` serta `java.net` juga mengalami peningkatan. Di paket `java.io` ditambahkan kelas baru bernama `Console`. Selain itu, ada sedikit modifikasi dan penambahan di kelas-kelas dan interface-interface paket `java.io`. Sementara itu, di paket `java.net` ditambahkan sebanyak dua interface dan empat kelas baru.

- Web Services

Untuk mendukung integrasi web services di edisi standard (J2SE), Java menambahkan API baru JAX-WS. Sebenarnya, dukungan Java

terhadap web services bukan merupakan hal baru karena sudah diimplementasikan di edisi enterprise (J2EE).

- Scripting

Java 6 menambahkan API scripting yang diimplementasikan melalui paket `javax.script`. Paket ini terdiri atas kelas-kelas dan interface-interface yang mendefinisikan engine scripting dan menyediakan framework untuk penggunaannya di aplikasi-aplikasi Java. API ini dimaksudkan untuk digunakan oleh pemrogram yang ingin mengeksekusi program-program yang ditulis dengan bahasa script di aplikasi Java.

- Sekuriti dan Performansi

Java 6 mencoba menyederhanakan tugas administrator sekuriti dengan menyediakan berbagai pendekatan baru untuk mengakses layanan sekuriti native, seperti *Public Key Infrastructure* (PKI) dan layanan kriptografi di Microsoft Windows, *Java Generic Security Services* (Java GSS) dan layanan Kerberos, dan akses ke server LDAP untuk autentikasi pengguna.

Untuk memperlihatkan tingkat kematangannya, Java 6 melakukan evaluasi serta peningkatan performansi secara menyeluruh. Dalam konteks aplikasi GUI Swing misalnya, keberadaan objek `SwingWorker` secara signifikan mampu meningkatkan performa aplikasi.